iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 11 **Issue:** I **Month of publication:** January 2023

DOI: https://doi.org/10.22214/ijraset.2023.48690

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue I Jan 2023- Available at www.ijraset.com

Generating Bipedal Pokémon Images by Implementing the Wasserstein Generative Adversarial Network

Jacqueline Jermyn
Department of Electrical and Computer Engineering, Florida Agricultural and Mechanical University-Florida State University College of Engineering, Tallahassee, Florida, United States of America

Abstract: ***Pokémon is a video game series wherein players capture and train fauna that are known as Pokémons. These creatures vary in colour, shape, size, and have distinct personalities and skillsets. After the Pokémons are trained by their coaches (who are the players), they are used to fight other Pokémons so that the winning player could achieve the Pokémon Champion status. Since its debut in 1996, Pokémon has become the highest grossing media franchise with over $100 billion in revenue through the sale of books and merchandise along with television programming and movies. This paper discusses the implementation of the Wasserstein Generative Adversarial Network (WGAN) to create artificial bipedal Pokémon images. This is achieved by training a WGAN on a custom-built training dataset of Generation I bipedal Pokémon pictures. The effectiveness of the WGAN for outputting artificial Pokémon images is evaluated by using the loss curves and by visually inspecting the outputted images.***
Keywords: Generative models, Generative Adversarial Networks (GAN), Wasserstein GAN (WGAN), Pokémon, loss curves, generator, critic, training dataset

I. INTRODUCTION

Generative models are unsupervised machine learning methods that model inputted data to produce artificial datapoints [1]. Generative Adversarial Networks (GANs) are generative models in which two sub-networks are trained concurrently but against each other to make artificial datapoints that resemble training datapoints [2]. For a GAN, these sub-networks are a generator and a discriminator, whereas for a Wasserstein GAN (WGAN) they are a generator and a critic [2-3]. GANs have many applications to include generating artificial audio, video, and images [4]. GANs could enhance music production by generating other types of sounds that could be incorporated into a music composition [5]. They could also produce realistic artificial videos for content in movies and television shows [6]. Furthermore, GANs could create graphic art by being trained on images that resemble the desired output [7]. This paper discusses the implementation of WGAN to generate artificial images of Generation I bipedal Pokémons by training it on a custom-built dataset of Pokémon images. The effectiveness of the WGAN technique is evaluated by reviewing the generator and critic loss curves and by examining the outputted images.

The rest of this paper is structured as follows: Section II provides background material on the WGAN technique. Section III describes the experimental procedures, and Section IV reviews the results. Section V discusses the conclusions and future work.

II. BACKGROUND INFORMATION

Wasserstein GAN (WGAN) is a variation of GAN that prevents mode collapse, while improving training stability [3,8]. Mode collapse occurs when there is little variation in the outputted artificial images [9]. Training could become unstable as the generator and the discriminator are trained simultaneously, but they are trained against each other. Hence, as the generator performance improves, the discriminator performance degrades. Training stability occurs when these two networks reach a Nash equilibrium through the optimization of the loss functions that compare the inputted and outputted datapoints [10-11]. The GAN discriminator loss function is based on the Jensen-Shannon (JS) divergence. This metric calculates the similarity between two probability distributions [12]. The equations for calculating the JS divergence, $D_{JS}(g||h)$, are provided as follows:

$$D_{KL}(g||h) = -\sum_{x \in X} g(x) \log\left(h(x)/g(x)\right) \quad (1)$$

$$D_{JS}(g||h) = \frac{1}{2} D_{KL}\left(g \| \frac{g+h}{2}\right) + \frac{1}{2} D_{KL}\left(h \| \frac{g+h}{2}\right) \quad (2)$$

where $D_{KL}(g||h)$ is the Kullback–Leibler divergence between distribution g and distribution h, x is a random event, and X is the sample space [13-15]. The discriminator's loss function could cause instability because the JS divergence saturates when the discriminator improves at differentiating between artificial and training dataset images resulting in a vanishing gradient error [12]. The gradients become stuck at zero, and the generator is unable to produce artificial images [9,12,16].

The WGAN technique improves training stability through the critic's loss function that is based on the Wasserstein metric [8-9]. It calculates the cost of changing the shape of one probability distribution into the shape of another [15,17]. The equation for this metric, $W(g,h)$, is provided as follows:

$$W(g,h) = \inf_{\gamma \sim \prod(g,h)} \mathbb{E}_{(x,y)\sim\gamma}[\|x-y\|] \quad (3)$$

where g and h are probability distributions, and $\prod(g, h)$ is the set of all joint probability distributions over x and y. In addition, γ is the amount of mass that is required to be moved from x to y to transform probability distribution g into probability distribution h. The infimum (inf) operator represents that the Wasserstein metric yields the lowest cost [12]. Because calculating the Wasserstein metric using equation 3 is intractable, it is approximated as follows:

$$W(g,\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim g}[f(x)] - \mathbb{E}_{x\sim\theta}[f(x)] \quad (4)$$

where θ is a probability distribution that approximates distribution h. The supremum (sup) operator is the least upper bound of the subset, and it is taken over all 1-Lipschitz functions [18-19]. The gradients of the critic network are also clipped to a fixed range of values to enforce this Lipschitz constraint [18,20]. The Wasserstein distance metric is more effective because it is a continuously differentiable function [12]. The first order derivative is continuous everywhere, which prevents the gradients of the WGAN critic from becoming stuck at zero [21-22].

The structure of the WGAN consists of two subnetworks: a generator and a critic. They sequentially process input data. The generator typically has an input layer, a dense layer, batch normalization (BN) layers, Leaky rectified linear unit (ReLU) activation functions, a reshape layer, Conv2DTransponse layers, and a hyperbolic tangent (tanh) activation function. The BN layers normalize the outputs of the dense layer and of the Conv2DTranspose layers [23-24]. The Leaky ReLU activation functions scale the outputs of the BN layers. When positive values are inputted to the Leaky ReLU activation function, they remain the same, whereas when negative values are inputted, they are scaled to small negative values [25]. The reshape layer restructures the output of the first Leaky ReLU activation function from a vector to a tensor. The Conv2DTranspose layers pass sliding windows over their inputs to create image features [26-27]. They have three inputs: the number of filters, the kernel size, and the stride size. The number of filters refers to the number of sliding windows, the kernel size specifies the dimensions of each sliding window, and the stride specifies the distance each sliding window moves each time it is passed over the input [26,28]. The last layer is the tanh activation function that scales inputted values to values between negative and positive one [29]. The final output of the WGAN generator is an artificial image. The structure for the WGAN generator is provided in Fig. 1

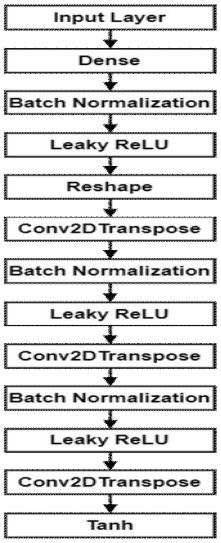


Fig. 1 WGAN Generator Structure

The critic compares the distribution of the training images with the distribution of the artificial images [12]. It consists of an input layer, Conv2D layers, Leaky ReLU activation functions, dropout, flatten, and dense layers. The input could be either an artificial image or a training dataset image. The Conv2D layers pass sliding windows over the input to analyse the features of the inputted image, but they perform the opposite operation of the Conv2DTranspose layers in the generator. The Conv2D layers have parameters that pertain to the number of filters, the kernel size, and the stride size [30]. The Leaky ReLU activation functions scale the outputs of the Conv2D layers. The dropout layers randomly set some of the outputs from the Leaky ReLU activation functions to zero to prevent overfitting [31-32]. The flatten layer converts the output of the last dropout layer to a vector. The dense layer ensures that the output is a single value that scores whether an image is from the training dataset. Values closer to negative one correspond to images from the training dataset, while values closer to positive one correspond to artificial images [33]. The structure for the WGAN critic is provided in Fig. 2.

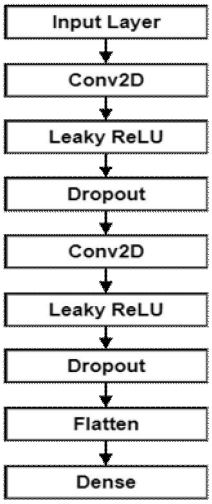


Fig. 2 WGAN Critic Structure

III.EXPERIMENTAL PROCEDURE

This investigation implemented the WGAN to generate artificial Pokémon images based on four Generation I bipedal Pokémon characters. Four Pokémon images that are available in the public domain were downloaded to develop a training dataset to train the WGAN [34-38]. They included Charmander, Charmeleon, Squirtle, and Wartortle. Charmander is mellow apricot with a linen underside. It has a smooth rounded head with blue eyes. The tip of its tail is on fire [34]. The picture of Charmander is provided in Fig. 3.

Fig. 3 Charmander [34-35]

Charmeleon is orange red with a Givry underside. It has a horned angular head. It has blue eyes, and the tip of its tail is also on fire [36]. Its image is shown in Fig. 4.

Fig. 4 Charmeleon [35-36]

Squirtle and Wartortle resemble turtles. Squirtle has crystal blue skin. Its shell has an alloy orange upper side with a Crayola's yellow underside. It has a smooth rounded head with brown eyes. Squirtle has a crystal blue tail with one black swirl [37]. The picture of Squirtle is provided in Fig. 5.

Fig. 5 Squirtle [35,37]

Wartortle is Jordy blue. Its shell has a medium wood topside with a Hampton underside. It has a rounded head with two feather-like ears, and it has brown eyes. It also has a long feather-like tail with two swirls [38]. Its image is shown in Fig. 6.

Fig. 6 Wartortle [35,38]

A Python program was written to create a training dataset with 1000 images based on these four Pokémons. These images had dimensions of 64x64-pixel. The first step initialized the input variables to include the name of the folder with the four original Pokémon pictures and the name of the folder for the new dataset. A list of the file paths for these four Pokémon images was then obtained. The next step was to select 996 file paths with replacement from the list of the four original image file paths. These file paths were also saved to a list. The first image in the list of file paths for the dataset was then copied to the dataset folder. It was given a unique numerical filename to prevent conflicts when additional images were copied to the dataset folder.

Random transformations, to include changes in colour level, contrast, and sharpness, were applied to each picture. The colour level affects the amount of colour in an image, and it varies between zero and one. A colour level of zero corresponds to a black and white image, while a colour level of one corresponds to the original image [39]. The contrast is the range of brightness values of a picture. A contrast level of zero corresponds to a solid grey picture, while a value of one corresponds to the original picture [40]. The sharpness affects the blurriness of an image, and it varies between zero and two [40-41]. A value of zero corresponds to a blurry picture, a value of one corresponds to the original picture, and a value of two corresponds to a sharpened image [40]. The program then determined if all 996 images were copied to the dataset folder. If not, each image was copied to the dataset folder, renamed, and transformed until all images were copied. The program also copied the four original Pokémon images to the dataset folder so that there were a total of 1000 images in this dataset. The program then terminated. The activity diagram for the Python program to create the training dataset is provided in Fig. 7.

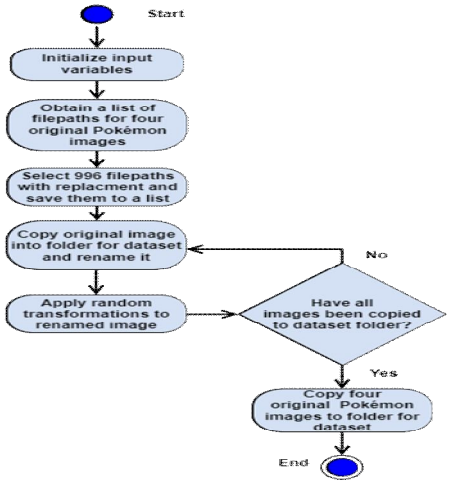


Fig. 7 Program for Creating Training Dataset

A Python program was written to train the WGAN on the custom-built dataset to create artificial Pokémon images. The first step was to initialize the input variables to include the maximum number of epochs, the batch size, and the noise dimension size. The maximum number of training epochs was 200 to prevent overfitting. The batch size was 64 because using a larger batch size could cause difficulties with convergence [42]. The noise dimension size is the number of elements of the random noise vectors. The default noise vector size of 100 elements was used for this implementation [42-43]. Next, the model architectures and hyperparameters for the generator and for the critic were defined. It involved specifying the layers and the hyperparameters for each layer. For this implementation, the default hyperparameters for each layer were used [43]. The hyperparameters for the generator and for the critic are provided in Tables I and II respectively. For these tables, N/A indicates that a layer does not have hyperparameters.

TABLE I
GENERATOR HYPERPARAMETERS

Layer	**Hyperparameters**
Input	units: 100
Dense	units: 65536
BN	N/A
Leaky ReLU activation	N/A
Reshape	16x16x256
Conv2DTranspose	filters: 128, kernel: 5x5, stride: 1x1
BN	N/A
Leaky ReLU activation	N/A
Conv2DTranspose	filters: 64, kernel: 5x5, stride: 2x2
BN	N/A
Leaky ReLU activation	N/A
Conv2DTranspose	filters: 3, kernel: 5x5, stride: 2x2
Tanh activation	N/A

TABLE III
CRITIC HYPERPARAMETERS

Layer	Hyperparameters
Input	Input shape: 64x64x3
Conv2D	filters: 64, kernel: 5x5, strides: 2x2
Leaky ReLU activation	N/A
Dropout	0.3
Conv2D	filters: 128, kernel: 5x5, strides: 2x2
Leaky ReLU activation	N/A
Dropout	0.3
Flatten	N/A
Dense	units: 1

The program began the first training epoch by loading a batch of training dataset images from their image files. They were inputted to the critic to score them. A batch of 64 random noise vectors was created and inputted to the generator to produce a batch of artificial images. These artificial images were also inputted to the critic to score them. The generator and the critic loss function values were calculated. The equations for the generator and for the critic loss functions are provided as follows:

$$L_{generator} = D(G(z))$$

(5)

$$L_{critic} = D(x) - D(G(z))$$

(6)

where $L_{generator}$ is the generator loss function, L_{critic} is the critic loss function, z is a random noise vector, $G(z)$ is the output of the generator for noise vector z, $D(G(z))$ is the critic's score for the output of the generator, and $D(x)$ is the critic's score for a training datapoint [44].

The gradients for the generator and for the critic were then computed. The critic gradients were clipped to a range of -0.01 to 0.01. Gradient values that were less than -0.01 were set to -0.01, values that were between -0.01 and 0.01 were unchanged, and values that were greater than 0.01 were set to 0.01. The generator and the critic training weights were then updated. The program then determined if all batches of training dataset images were loaded. If this was not the case, additional batches were loaded and the WGAN was trained on them until all batches were loaded to complete this training epoch. This program next determined if the specified number of training epochs were completed. If not, the WGAN was trained for additional training epochs until the specified number of epochs were finished. The program then outputted the final images of artificial Pokémon and terminated. The activity diagram for the WGAN program is provided in Fig. 8.

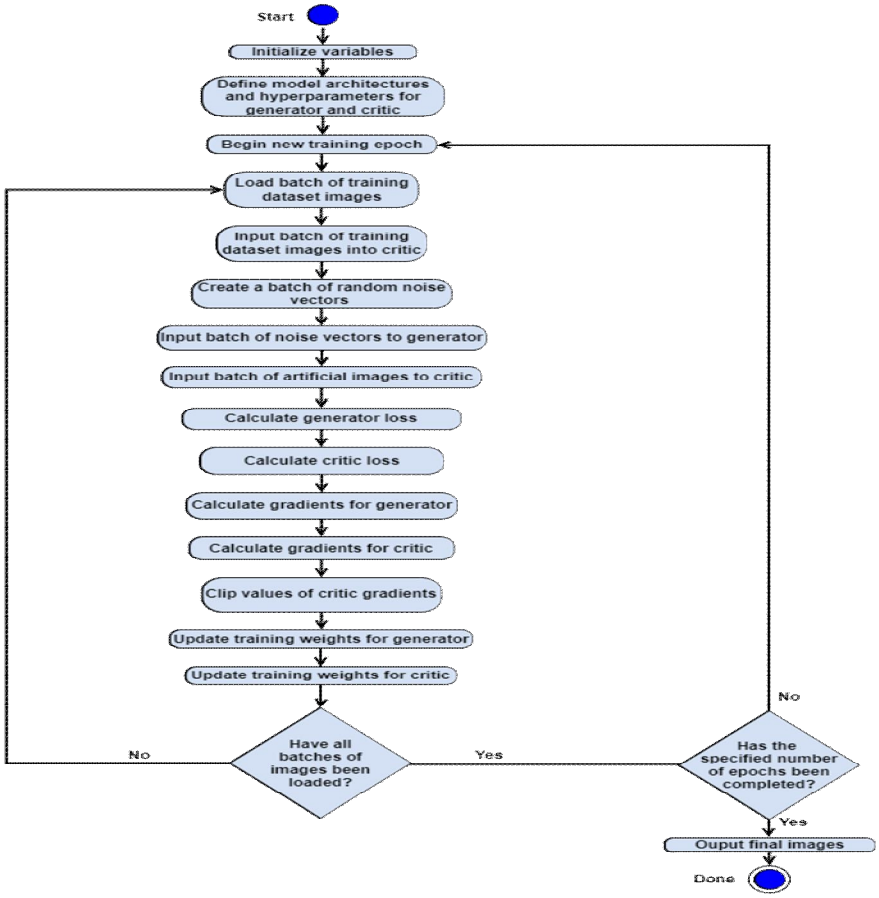


Fig. 8 WGAN Activity Diagram

The loss curves for the generator and for the critic were plotted to provide training stability information and also to provide an indication as to when training should be terminated to prevent overfitting.

IV. EXPERIMENTAL RESULTS

Two factors affect when to stop training: image quality and stabilization of the loss curves. Image quality is the more important of these factors. Although the loss curves may show that training stability has been achieved, training should continue until the desired image quality is reached. For this study, the generator and the critic loss curves are provided in Fig. 9.

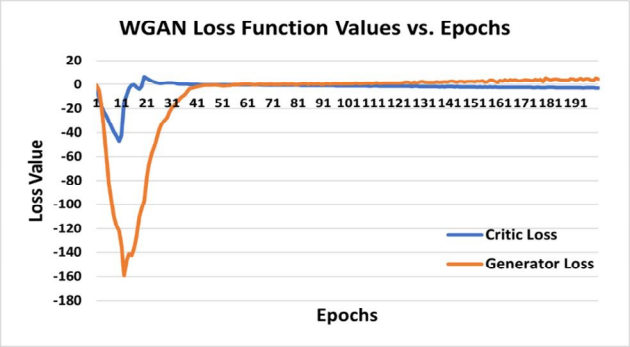


Fig. 9 Loss Curves for Generator and for Critic

These loss curves show that initially, the loss curves for the generator and for the critic plunged rapidly. As training continued, both loss curves rose. Training becomes stable when both loss curves converge to zero. For this study, training became stable after 60 epochs. A review of the images at 60 epochs showed that the WGAN generated new Pokémon images that incorporated characteristics of Pokémons from the training dataset. However, image quality was substandard because they were blurry. These images are shown in Fig. 10.

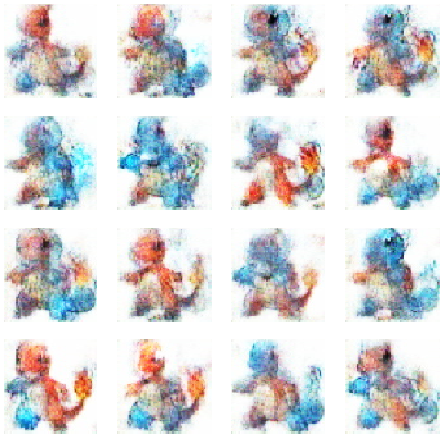


Fig. 10 Artificial Pokémon Images at 60 Epochs

The outputted Pokémon images at 100 epochs were still blurry, but they incorporated features from various Pokémons from the training dataset. These images are shown in Fig. 11.

Fig. 11 Artificial Pokémon Images at 100 Epochs

The outputted Pokémon images at 150 epochs show an improvement in the image quality. Training was also stable as these images were different than the training dataset images. The outputted images are shown in Fig. 12.

Fig 12. Artificial Pokémon Images at 150 Epochs

The pictures that were outputted at 200 epochs were much higher in quality, but overfitting was beginning to occur. Two of the outputted images resembled the original Charmeleon and Squirtle images. A review of the loss curves also confirmed that instability was beginning to occur because the generator loss curve was rising again. Training should therefore be terminated at 200 epochs. These outputted images are shown in Fig. 13.

Fig 13. Artificial Pokémon Images at 200 Epochs

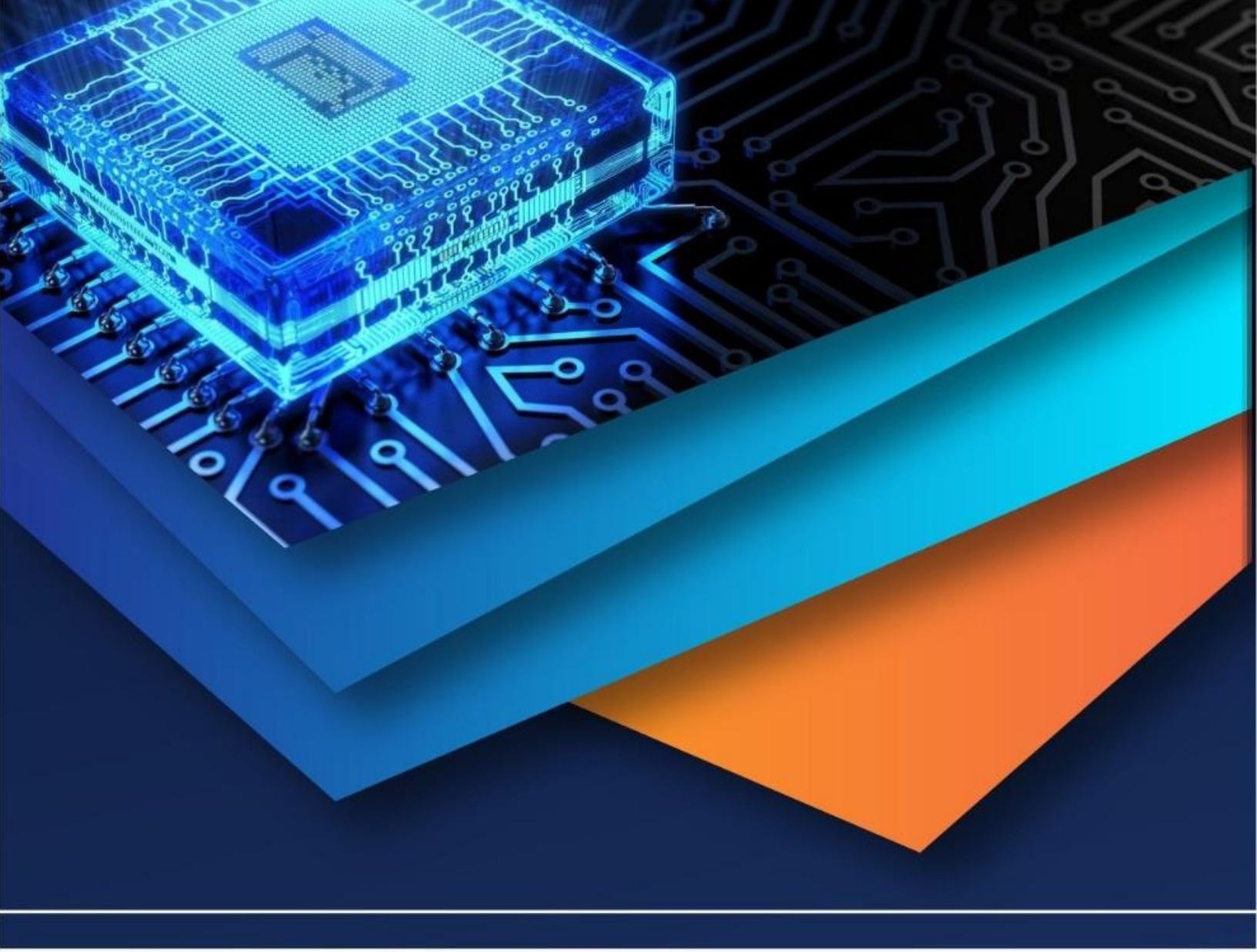
V. CONCLUSIONS AND FUTURE WORK

This investigation implemented the WGAN to output images of bipedal Pokémon by training this generative model on a custom-built training dataset of Generation I bipedal Pokémon pictures. The efficacy of this technique was determined by using the loss curves for the generator and for the critic and by visually examining the outputted Pokémon images. Results showed that WGAN successfully created pictures of artificial Pokémon. Furthermore, the loss curves illustrated that training stabilized at 60 epochs, but the outputted images were blurry. Because of this reason, training was continued for 100, 150, and 200 epochs. The images outputted at 150 epochs were sharper than those produced at 60 and at 100 epochs, and training was still stable. The Pokémon images that were generated at 200 epochs were the highest quality, but overfitting was beginning to take place as two of the outputted images were the same as the training dataset images. Because of this reason, training was halted at 200 epochs. Future work could involve implementing least squares GAN, Fisher GAN, and Cramer GAN to generate Pokémon images [45].

REFERENCES

[1] "Background: What is a Generative Model?" Developers.google.com. https://developers.google.com/machine-learning/gan/generative (accessed Jan. 9, 2023).
[2] J. Brownlee. "A Gentle Introduction to Generative Adversarial Networks (GANs)." Machinelearningmastery.com. https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/ (accessed Jan. 9, 2023).
[3] M. Arjovsky, S. Chintala, L. Bottou, "Wasserstein GAN," 2017, *arXiv:1701.07875*.
[4] M. Raj. "Face Generation Using Generative Adversarial Networks (GAN)." Medium.com. https://medium.com/nerd-for-tech/face-generation-using-generative-adversarial-networks-gan-6d279c2d5759 (accessed Jan. 9, 2023).
[5] "Train Generative Adversarial Network (GAN) for Sound Synthesis." Mathworks.com. https://www.mathworks.com/help/audio/ug/train-gan-for-sound-synthesis.html (accessed Jan. 9, 2023).
[6] N. Aldausari, A. Sowmya, N. Marcus, and G. Mohammadi, "Video Generative Adversarial Networks: A Review," 2020. [Online]. Available: arXiv:2011.02250v1.
[7] N. Pirani. "How to create fake images on a computer using GAN's." Medium.com. https://nylapirani.medium.com/how-to-create-fake-images-on-a-computer-using-gans-9ef75d6cb85d (accessed Jan. 9, 2023).
[8] "Wasserstein GAN." Paperswithcode.com. https://paperswithcode.com/method/wgan (accessed Jan. 6, 2023).
[9] "Common Problems." Developers.google.com. https://developers.google.com/machine-learning/gan/problems (accessed Jan. 9, 2023).
[10] J. Brownlee. "Tips for Training Stable Generative Adversarial Networks." Machinelearningmastery.com. https://machinelearningmastery.com/how-to-train-stable-generative-adversarial-networks/ (accessed Jan. 12, 2023).
[11] O. Nachum. "What are Generative Adversarial Networks (GANs)?" Quora.com. https://www.quora.com/What-is-meant-by-Generative-Adversarial-Networks-GANS-have-stability-problems-during-their-training-phase (accessed Jan. 12, 2023).
[12] A. Sankar. "Demystified: Wasserstein GANs (WGAN)." Towardsdatascience.com. https://towardsdatascience.com/demystified-wasserstein-gans-wgan-f835324899f4 (accessed Jan. 9, 2023).
[13] J. Brownlee. "How to Calculate the KL Divergence for Machine Learning." Machinelearningmastery.com. https://machinelearningmastery.com/divergence-between-probability-distributions/ (accessed Jan. 12, 2023).
[14] S. Yadav. "Necessary Probability Concepts for Deep Learning: Part 1." Medium.com. https://medium.com/@sunil7545/necessary-probability-concepts-for-deep-learning-557f75dd3bce (accessed Jan. 12, 2023).
[15] S. Yadav. "Necessary Probability Concepts for Deep Learning: Part 2." Medium.com. https://medium.com/@sunil7545/kl-divergence-js-divergence-and-wasserstein-metric-in-deep-learning-995560752a53 (accessed Jan. 12, 2023).
[16] "Monitor GAN Training Progress and Identify Common Failure Modes." Mathworks.com. https://www.mathworks.com/help/deeplearning/ug/monitor-gan-training-progress-and-identify-common-failure-modes.html (accessed Jan. 12, 2023).
[17] B. Fisher. "The Earth Mover's Distance." Homepages.inf.ed.ac.uk. https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/RUBNER/emd.htm (accessed Jan. 6, 2023).
[18] N. Adaloglou. "How to stabilize GAN training." Towardsdatascience.com. https://towardsdatascience.com/wasserstein-distance-gan-began-and-progressively-growing-gan-7e099f38da96 (accessed Jan. 13, 2023).
[19] A. Irpan. "Read-through: Wasserstein GAN." Alexirpan.com. https://www.alexirpan.com/2017/02/22/wasserstein-gan.html (accessed Jan. 13, 2023).
[20] "What is Lipschitz constraint and why it is enforced on discriminator?" Ai.stackexchange.com. https://ai.stackexchange.com/questions/29904/what-is-lipschitz-constraint-and-why-it-is-enforced-on-discriminator (accessed Jan. 13, 2023).
[21] "Definition: Continuously Differentiable." Proofwiki.org. https://proofwiki.org/wiki/Definition:Continuously_Differentiable (accessed Jan. 13, 2023).
[22] J. Hui. "GAN – Wasserstein GAN & WGAN-GP." Medium.com. Jonathan-hui.medium.com. https://jonathan-hui.medium.com/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490 (accessed Jan. 13, 2023).
[23] R. Alake. "Batch Normalization in Neural Networks Explained (Algorithm Breakdown)." Towardsdatascience.com. https://towardsdatascience.com/batch-normalization-explained-algorithm-breakdown-23d2794511c (accessed Jan. 9, 2023).
[24] J. Huber. "Batch Normalization in 3 Levels of Understanding." Towardsdatascience.com. https://towardsdatascience.com/batch-normalization-in-3-levels-of-understanding-14c2da90a338 (accessed Jan. 9, 2023).
[25] J. Brownlee. "A Gentle Introduction to the Rectified Linear Unit (ReLU)." Machinelearningmastery.com. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/ (accessed Jan. 9, 2023).
[26] K. Wei. "Understand Transposed Convolutions." Towardsdatascience.com. https://towardsdatascience.com/understand-transposed-convolutions-and-build-your-own-transposed-convolution-layer-from-scratch-4f5d97b2967 (accessed Jan. 9, 2023).
[27] K. Shahhosseini. "In Keras what is the difference between Conv2DTranspose and Conv2D." Stackoverflow.com. https://stackoverflow.com/questions/68976745/in-keras-what-is-the-difference-between-conv2dtranspose-and-conv2d (accessed Jan. 10, 2023).
[28] "tf.keras.layers.Conv2DTranspose." tensorflow.org. https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2DTranspose (accessed Jan. 10, 2023).
[29] S. Sharma. "Activation Functions in Neural Networks." Towardsdatascience.com. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 (accessed Jan. 10, 2023).
[30] A. Rosebrock. "Keras Conv2D and Convolutional Layers." Pyimagesearch.com. https://pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/ (accessed Jan. 10, 2023).
[31] "Dropout Layer." Keras.io. https://keras.io/api/layers/regularization_layers/dropout/ (accessed Jan. 10, 2023).
[32] C. Goyal. Analyticsvidhya.com. https://www.analyticsvidhya.com/blog/2021/06/complete-guide-to-prevent-overfitting-in-neural-networks-part-2/ (accessed Jan. 10, 2023).
[33] J. Brownlee. "How to Develop a Wasserstein Generative Adversarial Network (WGAN) From Scratch." Machinelearningmastery.com. https://machinelearningmastery.com/how-to-code-a-wasserstein-generative-adversarial-network-wgan-from-scratch/ (accessed Dec. 31, 2022).
[34] "Charmander (Pokémon)." Bulbapedia.bulbagarden.net. https://bulbapedia.bulbagarden.net/wiki/Charmander_(Pok%C3%A9mon) (accessed Jan. 10, 2022).
[35] Kvpratama. "Pokémon Images Dataset." Kaggle.com. https://www.kaggle.com/datasets/kvpratama/pokemon-images-dataset (accessed Jan. 10, 2022).
[36] "Charmeleon (Pokémon)." Bulbapedia.bulbagarden.net. https://bulbapedia.bulbagarden.net/wiki/Charmeleon_(Pok%C3%A9mon) (accessed Jan. 10, 2022).

[37] "Squirtle (Pokémon)." Bulbapedia.bulbagarden.net. https://bulbapedia.bulbagarden.net/wiki/Squirtle_(Pok%C3%A9mon) (accessed Jan. 10, 2022).
[38] "Wartortle (Pokémon)." Bulbapedia.bulbagarden.net. https://bulbapedia.bulbagarden.net/wiki/Wartortle_(Pok%C3%A9mon) (accessed Jan. 10, 2022).
[39] "Image Enhancement in PIL." Geeksforgeeks.org. https://www.geeksforgeeks.org/image-enhancement-in-pil/ (accessed Jan. 10, 2022).
[40] "ImageEnhance Module." Pillow.readthedocs.io. https://pillow.readthedocs.io/en/stable/reference/ImageEnhance.html (accessed Jan. 11, 2022).
[41] P. Jirsa. "The Ultimate Guide to Contrast in Photography." Adorama.com. https://www.adorama.com/alc/the-ultimate-guide-to-contrast-in-photography/ (accessed Jan. 11, 2022).
[42] T. Mittal. "Tips On Training Your GANs Faster and Achieve Better Results." Medium.com. https://medium.com/intel-student-ambassadors/tips-on-training-your-gans-faster-and-achieve-better-results-9200354acaa5 (accessed Jan. 14, 2023).
[43] Cousin_Zan. "WGAN tf2." Kaggle.com. https://www.kaggle.com/code/cousinzan/wgan-tf2 (accessed Jan. 14, 2023).
[44] "Loss Functions." Developers.google.com. https://developers.google.com/machine-learning/gan/loss (accessed Jan. 14, 2023).
[45] H. Heidenreich. "GAN Objective Functions: GANs and Their Variations." Towardsdatascience.com. https://towardsdatascience.com/gan-objective-functions-gans-and-their-variations-ad77340bce3c (accessed Jan. 14, 2023).



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY